5

Objektorientierte kartographische Visualisierung

JÜRGEN DÖLLNER

5.1 Einführung

Dieses Kapitel stellt eine Reihe ausgewählter Software- und Visualisierungskonzepte für die Konstruktion von interaktiven und animierten kartographischen Visualisierungen vor. Der Bezug zur kartographischen Animation liegt darin, daß die auf diesen Konzepten basierende interaktive, dynamische Visualisierungsumgebung für die Herstellung von Echtzeit-Animationen verwendet werden kann. Die hier vorgestellten Konzepte verfolgen das Ziel, im Sinne des „map use cube" (MacEachren 1994) interaktive und perspektivische Karten zu ermöglichen, die für die intern monologische wie auch für die extern dialogische Nutzung geeignet sind, einen wahlweise niedrigen oder hohen Grad an Interaktivität bereitstellen und es erlauben, vorhandene Informationen zu präsentieren und Unbekanntes zu erforschen. Die Implementierung solcher Karten erweist sich als sehr komplex. Ausgefeilte computergraphische Verfahren, die Konzepte für Interaktions und Animation und das Software-Engineering stellen hohe Anforderungen, die mit Hilfe eines erweiterbaren, objektorientierten Visualisierungssystems bewältigt werden können. Kartographische Daten (zum Beispiel Geländedaten) werden zusammen mit ihren graphischen Repräsentationen als Software-Objekte in das Visualisierungssystem eingebunden.

Eine besondere Herausforderung für ein solches Visualisierungssystem liegt darin, leicht benutzbare und verständliche interaktive Darstellungsmöglichkeiten zu schaffen. Dazu müssen zum einen moderne Verfahren der Computergraphik zum Einsatz kommen, um eine Darstellung großer Datenmengen in Echtzeit zu gewährleisten. Zum anderen sind geeignete Metaphern für die Interaktion und Präsentation zu entwickeln, die von Anwendern einer solchen kartographischen Visualisierungsumgebung möglichst intuitiv verstanden und akzeptiert werden.

Für die Konstruktion einer kartographisch ausgerichteten Visualisierungsumgebung werden in diesem Beitrag folgende Konzepte und Verfahren aus der Computergraphik und dem Software-Engineering vorgestellt:

G. Buziek et al. (eds.), *Dynamische Visualisierung*

- Ansteuerung verschiedener Bilderzeugungssysteme (*rendering*);
- Modellierung virtueller 3D-Szenen basierend auf einer Graphendarstellung, die sowohl graphisch-geometrische als auch dynamische Aspekte virtueller 3D-Objekte berücksichtigt;
- Konzept zur Bildung von fachspezifischen Visualisierungskomponenten, den *3D-Widgets*.

Diese Software-Konzepte werden anhand einer Reihe konkreter kartographischer Visualisierungs- und Animationskomponenten vertieft. Das Kapitel schließt mit einer Bewertung der vorgestellten Konzepte, die aus den Erfahrungen bei der Implementierung einer interaktiven Visualisierungsumgebung für kartographische Anwendungen stammen.

5.2 Grundlagen der objektorientierten Visualisierung

Die objektorientierte Software-Entwicklung bietet die wesentlichen Voraussetzungen zur Konstruktion komplexer Software-Systeme, da mit ihrer Hilfe die Konzepte in einem Aufgabenbereich logisch in Module, den *Klassen* und *Paketen*, zerlegt und diese miteinander in Beziehung durch Generalisierung und Spezialisierung gebracht werden können. Grundlegendes Konstrukt ist die *Klasse*, die einen Objekttyp durch Spezifikation der *Klassenelemente*, das heißt durch Attribute und Methoden, beschreibt. Jede Klasse besitzt in Form seiner öffentlichen Klassenelemente eine klar definierte *Schnittstelle*, stellt Dienste in Form seiner *Methoden* nach außen bereit und verwaltet seine meist verborgenen *Attribute* im inneren. Eine Klasse kann die Eigenschaften einer anderen Klasse durch *Vererbung* übernehmen bzw. modifizieren. Die Wiederverwendbarkeit und Erweiterbarkeit derart aufgebauter Software ist deutlich höher als die von konventionell (zum Beispiel prozedural) entwickelter Software. Für eine Einführung in die objektorientierte Software-Entwicklung sei auf die kompakte Darstellung von Oestereich (1997) verwiesen. Objektorientierung hat sich im Bereich der Visualisierungs- und Computergraphik-Software fest etabliert und als erfolgreich erwiesen (Cunningham et al 1992).

5.2.1 Architektur eines interaktiven Visualisierungssystems

Die Software-Architektur eines interaktiven Visualisierungssystems läßt sich grob in drei Schichten einteilen:

- **3D-Rendering**: Diese Schicht enthält Verfahren und Techniken zur Bildsynthese. Sie unterscheiden sich hinsichtlich des Umfangs der verfügbaren Graphikprimitive und Graphikattribute, der Geschwindigkeit und der Qualität der Bilderzeugung. Zur Implementierung dieser Schicht wird im allgemeinen auf Standard-Software, zum Beispiel *OpenGL* (Woo et al 1997), zurückgegriffen.
- **3D-Modellierung**: Diese Schicht ist für die geometrische und gestalterische Modellierung virtueller Objekte und virtueller Szenen zuständig. Die Schicht ist im allgemeinen auch verantwortlich für die Spezifikation der Dynamik der

Objekte, das heißt ihres interaktiven und zeitlichen Verhaltens. *OpenInventor* (Wernecke 1994) und *Java 3D* (Brown und Petersen 1998, Mohan 1998) sind typische Vertreter von Graphiksystemen, die eine allgemeine Szenenmodellierung unterstützen.

- **3D-Komponenten**: Diese Schicht enthält häufig benötigte oder allgemein nützliche Anwendungsbausteine, die aus den Bestandteilen der Rendering-Schicht und der Modellierungsschicht gebildet werden. Diese Schicht vereinfacht damit die Entwicklung neuer Systeme durch vorgefertigte Komponenten mit meist komplexer Funktionalität.

Visualisierungssysteme setzen diese logische Gliederung nicht immer physikalisch um, das heißt die Schichten sind auf der Software-Ebene nicht wirklich getrennt, sondern ineinander verschränkt. In einem solchen Fall ist es im allgemeinen schwer, das Visualisierungssystem an spezifische Anwendungsbedürfnisse, zum Beispiel die der Kartographie, anzupassen, da jede Schicht gezielt um anwendungsspezifische Konstrukte erweitert werden muß.

Die objektorientierte Graphikbibliothek *VisualizationToolkit vtk* (Schroeder et al 1998) besteht aus einer expliziten Schicht für Modellierung und Komponenten, wohingegen die Rendering-Schicht nicht direkt zugänglich ist. *VRML (Virtual Reality Modeling Language)* kann vorwiegend als Modellierungsschicht betrachtet werden, da weder die Rendering-Schicht noch die Komponentenschicht festlegt ist (zu VRML siehe den Beitrag von Zedi). *OpenInventor* (Wernecke 1994) ist eine objektorientierte 3D-Graphikbibliothek, die analog zu VRML auf dem Szenengraphen-Paradigma basiert und Modellierungsfunktionalität bereitstellt. Sie arbeitet mit *OpenGL* als Rendering-Schicht und definiert keine explizite Komponentenschicht.

Java 3D (Mohan 1998), eine objektorientierte Graphikbibliothek für Java-Anwendungen und -Applets, bietet eine leistungsfähige Modellierungsschicht, jedoch keinen direkten Zugang zu der auf den einzelnen Hardware-Plattformen unterschiedlich realisierten Rendering-Schicht. Die Bibliothek unterstützt nicht explizit eine Komponentenschicht. Die in diesem Beitrag vorgestellten Konzepte wurden mit dem objektorientierten Visualisierungs- und Animationssystem *MAM/VRS* realisiert, der *Modeling and Animation Machine* und dem *Virtual Rendering System* (Döllner und Hinrichs 1997), dessen Schichten für Rendering, Modellierung und Komponenen logisch und physikalisch explizit modelliert sind.

5.2.2 Konzepte für Rendering-Schichten

Systeme für die Bilderzeugung (*rendering*) unterscheiden sich in Hinblick auf die verwendeten Rendering-Techniken und ihrer Schnittstelle zu Anwendungsprogrammen.

Rendering-Techniken

Bei den Rendering-Techniken repräsentiert das verwendete Beleuchtungsmodell ein entscheidendes Kriterium für die Charakteristik der generierten Bilder. Rendering-Techniken auf der Basis *lokaler Beleuchtungsmodelle* betrachten jedes Graphikobjekt bei der Schattierung isoliert, eine Wechselwirkung der Objekte untereinander wird nicht berücksichtigt. *OpenGL* ist ein typischer Vertreter eines Rendering-Systems für Echtzeit-Anwendungen mit lokalem Beleuchtungsmodell. OpenGL verwendet, wie viele andere Rendering-Systeme, das Beleuchtungsmodell von Phong und das Schattierungsverfahren von Gouraud (Foley et al 1994).

Rendering-Techniken auf der Basis *globaler Beleuchtungsmodelle* erzeugen Bilder, die einen höheren Grad an Realismus besitzen, da sie die Lichtverteilung präziser berechnen, etwa mit Hilfe des Ray-Tracing-Verfahrens oder des Radiosity-Verfahrens. Vertreter dieser Gruppe von Rendering-Systemen sind das Lichtsimulations-System *Radiance* (Ward und Shakespeare 1998) und das Ray-Tracing-System POV-Ray (Enzmann et al. 1994, siehe auch die Beiträge von Buziek, Dransch/Rase und Schiltz in diesem Buch). Eine einführende Darstellung von Rendering-Techniken geben Foley et al (1994).

Schnittstellen von Rendering-Systemen

Die Schnittstelle, die ein Rendering-System bereitstellt, besteht entweder aus einer Programmierschnittstelle, falls das Rendering-System im Sinne einer Softwarebibliothek in das Anwendungsprogramm eingebunden werden soll, oder ist durch eine Reihe von Formaten zur Datenübergabe mit einer Datei festgelegt. Allgemein besitzen heutige Rendering-Systeme eine Programmierschnittstelle (*application programming interface*, API) mit funktionsbasierter oder deklarativer Natur.

Ein *funktionsbasiertes API* wie zum Beispiel das von *OpenGL* und Microsofts *Direct3D* (Microsoft 1999) stellt die Funktionalität des Rendering-Systems über eine Reihe von Klassen oder Funktionen bereit, die ein Anwendungsprogramm zur Implementierung heranziehen kann. Ein solches API setzt voraus, daß der Anwendungsentwickler die Arbeitsweise und Struktur des Rendering-Systems kennt. Dadurch wird eine direkte und effiziente Nutzung aller Rendering-Fähigkeiten ermöglicht.

Ein *deklaratives API basiert* auf einer (meist dateibasierten) Szenenspezifikation, die vom Anwendungsprogramm explizit generiert und anschließend zur Bilderzeugung vom Rendering-System interpretiert wird, zum Beispiel von den Programmen POV-Ray oder Radiance. Die Rendering-Fähigkeiten solcher Systeme sind stets durch die Möglichkeiten der deklarativen Struktur beschränkt. Eine Erweiterung des Formats ist in der Regel nicht realisierbar. Rendering-Systeme mit diesem API können im allgemeinen nicht in ein Anwendungsprogramm integriert werden, sondern arbeiten autonom.

Rendering-Komponenten

Unabhängig von der Software-Schnittstelle und der Technik für die Bilderzeugung lassen sich eine Reihe von *Rendering-Komponenten* identifizieren und kategorisieren, über die jedes Rendering-System verfügt. Diese Aufteilung ist die Grundlage für eine generische Rendering-Schnittstelle, die zur Programmierung und Steuerung verschiedener Rendering-Systeme verwendet werden kann.

- **Geometrische Objekte**: Objekte dieser Kategorie spezifizieren die Geometrie von 2D- und 3D-Objekten, zum Beispiel Dreiecksnetze, Freiformflächen oder Linienzüge.
- **Graphische Attribute**: Objekte dieser Kategorie spezifizieren die Erscheinung von geometrischen Objekten, zum Beispiel Texturen, Materialien und Zeichenstile.
- **Geometrische Attribute**: Objekte dieser Kategorie transformieren bzw. manipulieren die Geometrie von Objekten, zum Beispiel Rotation, Translation, Skalierung und Clipping-Ebenen.
- **Rendering-Engines**: Objekte dieser Kategorie verwalten graphische und geometrische Attribute in einem Rendering-Kontext und übernehmen das eigentliche Rendering der Objekte. In der Implementierung einer generischen Rendering-Schnittstelle wird die Verwaltung und Auswertung auf das jeweilig zugrundeliegende konkrete 3D-Rendering-System abgebildet.

Das *Virtual Rendering System* (VRS) repräsentiert eine generische Rendering-Schnittstelle und erlaubt es einer Visualisierungsanwendung, das zugrundeliegende Rendering-System auszutauschen, ohne daß dafür der Anwendungscode geändert werden müßte (Döllner und Hinrichs 1999). Speziell für die Bedürfnisse der kartographischen Visualisierung ist diese Eigenschaft von Vorteil, weil für Planungs- und Explorationsaufgaben aufgrund der dafür notwendigen Interaktivität Echtzeit-Rendering-Systeme zum Einsatz kommen. Für Präsentationsaufgaben hingegen, zum Beispiel der Herstellung kartographischer Animationen, können photorealistische Bildfolgen ohne Entwicklungsmehraufwand erstellt werden.

5.2.3 Konzepte für Modellierungsschichten

Die Modellierungsschicht eines Visualisierungssystems übernimmt die Aufgabe, Szenen und die in ihnen dargestellten 2D- und 3D-Objekte zu spezifizieren. Der hier vorgestellte Ansatz ist eine verallgemeinerte Variante der Ansätze, die in den Software-Umgebungen Java 3D und VRML Eingang gefunden haben.

Eine Szenenspezifikation erfolgt in Form eines gerichteten, azyklischen Graphen, dem sog. *Szenengraphen*. Er beschreibt hierarchisch den geometrischen Aufbau einer Szene und legt die Attribute der einzelnen geometrischen Objekte fest. Die Knoten eines Szenengraphen sind mit geometrischen Objekten, graphischen Attributen und geometrischen Attributen verknüpft. Eine an den geometrischen und graphischen Attributen organisierte Hierarchie erlaubt die kompakte Darstellung gemeinsamer Attribute und insbesondere gemeinsamer lokale Koordinatensy-

steme einer Szene. Ein Beispiel eines einfachen Szenengraphen ist in Abbildung 5-1 illustriert. Er modelliert ein texturiertes, trianguliertes Geländemodell. Der Wurzelknoten enthält die Rendering-Komponenten, die die virtuelle Kamera spezifizieren. Zusätzlich zum Geländemodell sind zwei textuelle Markierungen im Szenengraphen beschrieben.

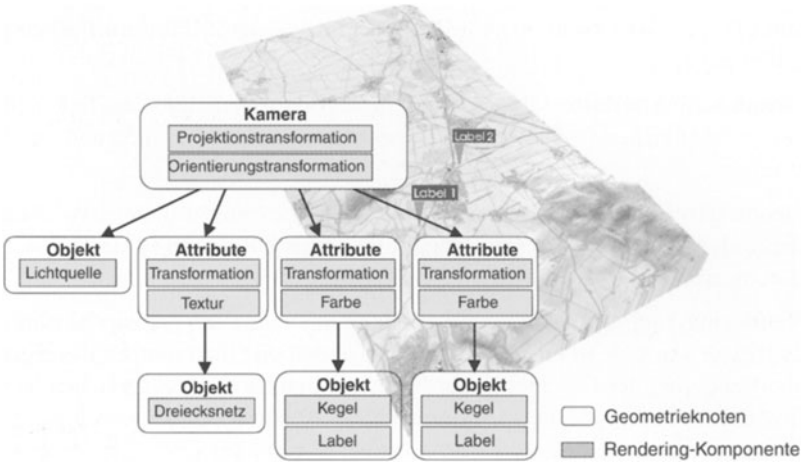


Abb. 5-1 Szenengraph zur Darstellung eines Geländemodells

Ein Mangel dieses Ansatzes liegt in der fehlenden Unterstützung der Spezifikation dynamischer Aspekte. Zeit- und ereignisabhängiges Verhalten kann nur schwer in eine Hierarchie eingefügt werden, die als Ordnungsprinzip den statischen-geometrischen Aufbau zugrunde liegen hat. Einige Konzepte, zum Beispiel *Java 3D*, *VRML* und *OpenInventor*, behelfen sich dadurch, daß an Knoten des Szenengraphen Sensoren installiert werden können, die Anwendungsfunktionen beim Eintreffen bestimmter Ereignisse aufrufen (*callbacks)*. Ein Sensor kann zum Beispiel bei Ablauf eines Zeitintervalls, bei Mausklick oder bei einem Fensterereignis reagieren, in dem er das dem Ereignis zugeordnete Callback auslöst.

Eine alternative Lösung stellt die symmetrische Modellierung von Geometrie und Verhalten dar. Analog zum Szenengraphen wird für eine virtuelle dreidimensionale Welt das Verhalten durch einen dem Szenengraph komplementär beigeordneten Verhaltensgraphen beschrieben. Die Knoten des Verhaltensgraphen spezifizieren das zeitliche oder ereignisabhängige Verhalten von 2D- und 3D-Objekten. Als Ordnungsprinzip wird der zeitliche Ablauf oder der Interaktionsablauf herangezogen.

In Abb. 5-2 ist ein Verhaltensgraph illustriert, der eine Kamerafahrt mit gleichzeitiger Bewegung einer Lichtquelle und eine Animation einer Textaufschrift (Label) beschreibt. Dazu werden einige der Rendering-Komponenten animiert, die im Szenengraphen (Abb. 5-1) ebenfalls verwendet werden. Die *sequentielle* Verhaltens-

gruppe beschreibt nacheinander ablaufende Handlungen; sie berechnet die Gesamtdauer der Animation aus der Summe der Dauer der Teilhandlungen. Die *simultane* Verhaltensgruppe beschreibt gleichzeitig ablaufende Handlungen. Die Gesamtdauer ergibt sich aus der längsten Dauer der Teilhandlungen. Verhaltensgraphen legen durch Zeitanforderungsknoten fest, wieviel „Lebenszeit" ihre Subgraphen benötigen. Verhaltensgruppen sammeln diese Anforderungen, errechnen daraus die Lebenszeiten ihrer Kindknoten, aktivieren Kindknoten, wenn deren Lebenszeit beginnt und deaktivieren Kindknoten, wenn ihre Lebenszeit endet. Zeit-Constraint-Knoten wenden zeitabhängige Funktionen, die von der Anwendung spezifiziert werden, auf Parameter der Rendering-Komponenten an. Ein Zeit-Constraint-Knoten modifiziert zum Beispiel die Orientierungstransformation der Kamera, um damit die Kameraposition zu animieren. Analog werden die Position der Lichtquelle und die Transformation des Labels mit Hilfe geeigneter zeitabhängiger Funktionen animiert.

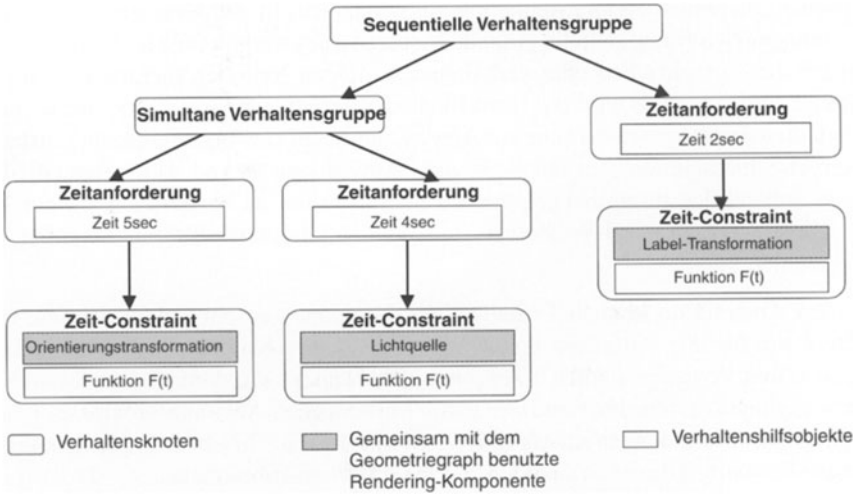


Abb. 5-2 Beispiel eines Verhaltensgraphen. Zeitabhängige Funktionen verändern Parameter assoziierter Rendering-Komponenten.

Szenengraphen und Verhaltensgraphen besitzen aus der Sicht der Rendering-Komponenten folgende Funktionen: der Szenengraph positioniert, orientiert und kombiniert die Rendering-Komponenten, wohingegen der Verhaltensgraph die Rendering-Komponenten manipuliert und animiert. Da die Rendering-Komponenten von Knoten beider Graphtypen assoziiert werden (die Rendering-Komponenten können gemeinsam von Knoten beider Graphtypen genutzt werden), lassen sich kompakt ereignis- und zeitabhängige Vorgänge formulieren und selbst komplexes zeitliches Verhalten objektorientiert mit Hilfe der Verhaltensknoten komponieren. Die hierarchische Anordnung von zeitbezogenen Verhaltensknoten erlaubt es, zeitliche Abläufe im Stil von Drehbüchern zu kodieren. Bei animierten und interaktiven 3D-Anwendungen kann beobachtet werden, daß Verhaltensgraphen und damit die Spezifikation der

Interaktion und der Animation einen weitaus größeren Teil der Programmspezifikation einnehmen als die Szenengraphen.

Die Modellierungsschichten der derzeitigen Visualisierungssysteme unterstützen im allgemeinen die Verhaltensmodellierung nur rudimentär, indem an Knoten des Szenengraphen anwendungsspezifische Funktionen geknüpft werden können, die beim Eintreffen bestimmter Ereignisse aufgerufen werden (Event-Callback-Mechanismus). Eine explizite Modellierung von Zeitflüssen wird in *MAM/VRS* durch eine Bibliothek leistungsfähiger Zeitknoten unterstützt (Döllner und Hinrichs 1997).

5.2.4 3D-Widgets als Visualisierungskomponenten

Die Entwicklung interaktiver, animierter Visualisierungen führt mit zunehmender Anwendungsgröße zu einer großen Anzahl von Graphen und Assoziationen zwischen Knotenobjekten und Rendering-Komponenten. In den Geometrie- und Verhaltensgraphen findet sich die gesamte Visualisierungsfunktionalität, die nach geometrischen, graphischen oder verhaltensabhängigen Kriterien hierarchisch angeordnet sind. Dadurch wird die Identifikation zusammengehöriger Geometrie- und Verhaltensknoten erschwert, die aus Anwendungssicht eine bestimmte, abgrenzbare Aufgabe übernehmen. Aus der Sicht einer Anwendung ist von daher eine Gliederung anhand der zusammengehörenden Funktionalität zu bevorzugen, da nur sie die Erweiterbarkeit und Wiederverwendung von komplexer Teilfunktionalität sicherstellen kann.

Zur Abstraktion können Visualisierungskomponenten entwickelt werden, die intern die für ihre Aufgaben notwendigen Rendering-Komponenten, Geometrieknoten und Verhaltensknoten erzeugen und verwalten. Mit derart konzipierten Visualisierungskomponenten und der damit verbundenen Abstraktion wird es möglich, große Anwendungen aus vorgefertigten Bausteinen effizient zu implementieren. Diese Bausteine können in Analogie zu Schnittstellen-Bibliotheken als *3D-Widgets* bezeichnet werden (Döllner und Hinrichs 1998). Sie ermöglichen es, anwendungsspezifische (zum Beispiel kartographische) Visualisierungstechniken wiederverwendbar und kompakt zu implementieren und in Bibliotheksform bereitzustellen. Sie erleichtern daher grundlegend die komponentenorientierte Entwicklung von Visualisierungsanwendungen.

3D-Widgets lassen sich formal als Muster von korrelierten Rendering-Komponenten, Geometriegraphen und Verhaltensgraphen definieren. Ein 3D-Widget kapselt im Sinne der objektorientierten Software-Entwicklung eine Menge von Rendering-Komponenten, Geometrieknoten und Verhaltensknoten (Abb. 5-3). Ein 3D-Widget legt durch seine Schnittstelle explizit fest, welche ihrer internen Objekte nach außen exportiert und welche Objekte von außen erwartet (das heißt importiert) werden. Der nachfolgende Abschnitt stellt Visualisierungskomponenten einer für geowissenschaftliche und kartographische Zwecke konzipierten 3D-Widget-Bibliothek vor.

5.3 Visualisierungskomponenten für Karten

In diesem Abschnitt wird erläutert, wie die bisher vorgestellten Konzepte eingesetzt werden können, um ein objektorientiertes kartographisches Visualisierungssystem zu konstruieren. Im Zentrum stehen dabei Komponenten zur Visualisierung von digitalen Geländemodellen (DGM), zur Navigation und Orientierung in Geländemodellen und zur Exploration und Analyse kartographischer Daten.

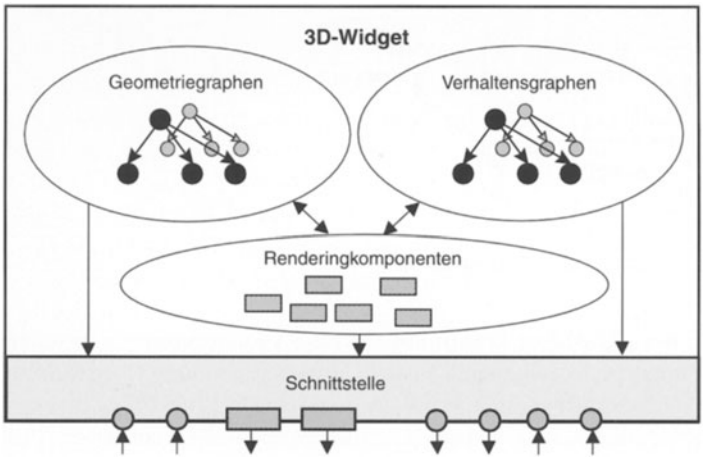


Abb. 5-3 Grundsätzlicher Aufbau eines 3D-Widgets

5.3.1 Software-Architektur

Die Architektur eines solchen Visualisierungssystems beruht auf dem Paradigma des *model view controller* (MVC, Abb. 5-4), das die Grundstruktur für alle Typen von graphischen Anwendungen widerspiegelt:

- Das Modell repräsentiert *Fachobjekte*, die im allgemeinen kartographische Daten enthalten. Fachobjekte sind in dem vorgestellten kartographischen Visualisierungssystem zum Beispiel digitale Geländemodelle (DGM), Landnutzungsinformationen und weitere thematische Daten. Die Fachobjekte stellen darüber hinaus Analysefunktionalität (zum Beispiel Abfragemethoden) bereit oder dienen als Mittler zu Geo-Informationssystem oder einer raumbezogenen Datenbank. Die Fachobjekte enthalten keine Visualisierungsfunktionen.
- Die Sichten (*views*) in Form von, zum Beispiel, perspektivischen Darstellungen eines Geländemodells sind zuständig für die Visualisierung des Modells, das heißt, sie übernehmen die Abbildung der Fachobjekte auf Rendering-Komponenten und Geometriegraphen. Dazu müssen insbesondere die in den Fachobjekten enthaltenen Daten effizient an die Rendering-Komponenten übertragen werden.

- Die *Controller* sind zuständig für die Manipulation des Modells. Sie kapseln Interaktionsstrategien und Animationsbeschreibungen, die durch Verhaltensgraphen implementiert werden können.

Das MVC-Paradigma bietet eine logische Gliederung für Visualisierungsanwendungen mit komplexen Visualisierungsstrategien. 3D-Widgets eignen sich zur Implementierung dieses Paradigmas, in dem sie die Funktionalität von Sichten und Controllern durch Geometriegraphen und Verhaltensgraphen implementieren.

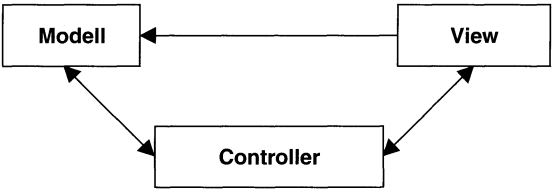


Abb. 5-4 Modell-View-Controller-Paradigma

Ein *kartographisches 3D-Widget* konstruiert für eine Visualisierungsanwendung Fachobjekte oder importiert sie von einer Visualisierungsanwendung (zum Beispiel DGM-Objekte), konstruiert Rendering-Komponenten (zum Beispiel Polygonnetze), konstruiert Geometriegraphen (zum Beispiel Geometrieknoten mit graphischen Attributen für Farbe, Material und Textur) und konstruiert Verhaltensknoten, welche die Dynamik der visualisierten Fachobjekte beschreiben (zum Beispiel Blickpunktauswahl in einem DGM durch Benutzerinteraktion). Für eine ausführliche Diskussion des MVC-Paradigms siehe Gamma et al (1996).

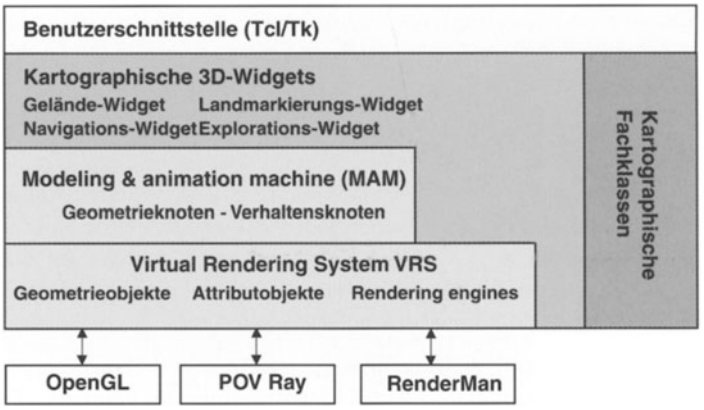


Abb. 5-5 Architektur des objektorientierten kartographischen Visualisierungssystems

Auf der Basis des Visualisierungs- und Animationssystems *MAM/VRS* wurde ein Prototyp eines kartographischen Visualisierungssystems entwickelt und erprobt, das auf den in den folgenden Abschnitten beschriebenen kartographischen 3D-Widgets basiert (Buziek und Döllner 1999). Die Systemarchitektur ist in Abb. 5-5 illustriert. Die kartographischen Fachklassen modellieren die kartographischen Daten. Die kartographischen 3D-Widgets sind sowohl an *MAM/VRS*-Komponenten als auch an kartographische Fachklassen gekoppelt. Die Benutzerschnittstelle bildet die oberste Systemschicht, die für die Konstruktion und Verwaltung der 3D-Widgets und der kartographischen Fachobjekte verantwortlich ist. Sie ist mit Hilfe der Skriptingsprache Tcl/Tk implementiert (Ousterhout 1994). Als Echtzeit-Rendering-System wird *OpenGL* eingesetzt. Zur Herstellung von tendenziell photorealistischen Bildern bzw. Filmen wird auf die Rendering-Systeme *POV-Ray* und *RenderMan* (Upstill 1990) zurückgegriffen.

5.3.2 Modellierung der kartographischen Darstellung digitaler Geländemodelle

Ein grundlegendes Fachobjekt stellt ein hybrides Geländemodell dar. Ein *Gelände-Widget* hat die Aufgabe, das Geländemodell in für die Visualisierung geeignete Rendering-Komponenten umzusetzen, die Schummerung des Geländes zu berechnen und die Texturierung zu steuern. Einige technische Aspekte dieses Widgets sind im folgenden beschrieben.

Generische Datenstruktur für die Echtzeit-Visualisierung

Ein hybrides Geländemodell wird in dem hier beschriebenen Ansatz durch Höhenwerte in einem regulären Gitter repräsentiert, die durch triangulierte Feinstrukturen ergänzt werden können (Buziek und Kruse 1992). Das Geländemodell wird weiter hierarchisch in verschiedenen Auflösungsstufen (*level of detail, LOD*) berechnet, die in einem sogenannten *Approximationsbaum* gespeichert werden. Dadurch kann bei der Bildgenerierung die Auflösung jedes Geländeteils je nach seiner Entfernung zur virtuellen Kamera gezielt gewählt werden. Dadurch läßt sich die Anzahl der Dreiecke, die zur Darstellung benötigt werden, reduzieren (*multiresolution modeling*). Die Datenstruktur ist generisch, das heißt, sie kann Geländedaten unterschiedlichen Typs in ein einheitliches LOD-Modell integrieren. Technisch erfolgt die Implementierung einerseits durch die in dem *virtual rendering system* gegebenen Rendering-Komponenten (zum Beispiel Dreiecksnetze), andererseits durch speziell für kartographische Anwendungen erstellte Geometrieobjekte. Die Auswahl der einzelnen LOD-Teilmodelle erfolgt bei der Bildgenerierung in Abhängigkeit einer Fehlerschranke für den Approximationsfehler. Für eine ausführliche Darstellung dieser Datenstruktur sei auf Baumann et al (1999) verwiesen.Gekoppelt an die geometrische Geländeinformation ist die thematische Information. Sie ist repräsentiert durch die auf das Gelände bezogenen Texturen. Die Texturen sind analog zum Geländemodell in einem hierarchischen LOD-Modell, dem *Texturbaum*, angeordnet.

Topographisches und thematisches Texturieren

Die Schummerung eines Geländemodells entsteht durch die im jeweiligen Rendering-System integrierte Schattierungsfunktionalität, die auf der Basis der geometrischen Darstellung und eines Beleuchtungsmodells Lichtintensitätswerte berechnet. Bei Echtzeit-Rendering-Systemen handelt es sich dabei meist um das Beleuchtungsmodell von Phong und das Schattierungsverfahren von Gouraud (Foley et al 1994). Bei einem LOD-Modell wechselt je nach Auflösungsgrad visuell deutlich wahrnehmbar die Schattierung, da die Farben nur für die Ecken eines Dreiecks berechnet und dann über das Dreieck interpoliert werden.

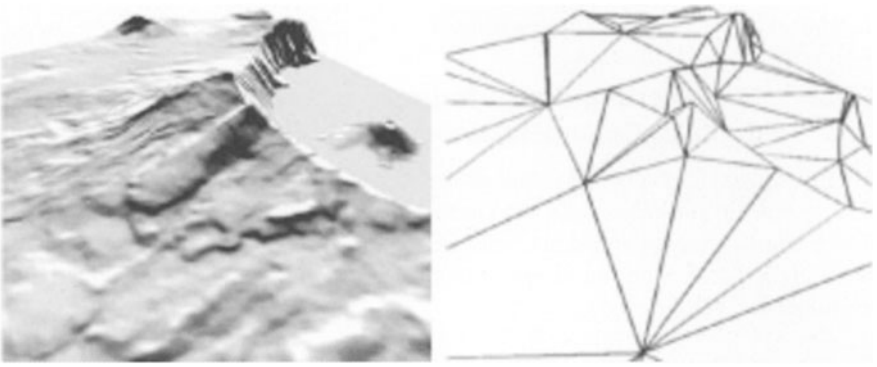


Abb. 5-6 Topographisch texturiertes Geländemodell (links), verwendete Triangulierung (rechts)

In dem vorgestellten kartographischen Visualisierungssystem wurde ein anderer Weg gewählt. Die gesamte Schattierung des Geländes wird vorausberechnet und in einer hoch aufgelösten 2D-Textur abgelegt. Diese Textur wird bei der Bilderzeugung dann auf das unbeleuchtete, geometrische Geländemodell projiziert. Dadurch ist die Schattierung unabhängig von der geometrischen Auflösung der einzelnen Geländeteile, da die Textur pixelbasiert ermittelt wird (Abb. 5-6). Sie ist allerdings abhängig von der Qualität der Textur-Projektion (mit bzw. ohne lineare Filterung) und der Texturgröße. Ein konstanter visueller Eindruck bleibt erhalten, wenn die geometrische Auflösung von Geländeteilen sich aufgrund von Kamerabewegungen ändert. Als Erweiterung können außerdem thematische Informationen bei der Herstellung der Geländetextur berücksichtigt werden. Diese Schattierungstechnik ist durch das in OpenGL verfügbare *Offscreen-Rendering* leicht umsetzbar, da die Schattentextur durch eine orthographische Geländeprojektion automatisiert und für den Benutzer unsichtbar im Hintergrund errechnet werden.

Außerdem können andere Schattierungsmodelle als die der gängigen Rendering-Systeme verwendet werden. So kann ein kartographisch bestimmtes Schattierungsmodell Bruchkanten berücksichtigten, die im geometrischen Modell nicht

oder nur bei bestimmten Auflösungsstufen enthalten sein würden. Die Qualität einer Geländemodelldarstellung hängt wesentlich von der in der Darstellung wahrnehmbaren Morphologie ab, die mit Hilfe spezialisierter Schummerungsmodelle präzise modelliert werden kann.

Darstellung farbiger Höhenschichten

Die aus der Kartographie bekannte Darstellung farbiger Höhenschichten durch höhenabhängige Färbung des Geländemodells kann durch das Gelände-Widget unter Ausnutzung eines effizienten Renderingverfahrens unterstützt werden. Dieses Verfahren nutzt eine Höhenskala, die die Höhenfarben enthält, und wendet diese als Textur implementierte Höhenskala auf das Gelände an, indem für jedes gezeichnete Pixel die entsprechende Farbe aus der Textur ausgewählt wird (Abb. 5-7). Die visuelle Darstellung ist damit pixelgenau und wird insbesondere von moderner PC-Graphikhardware effizient unterstützt.

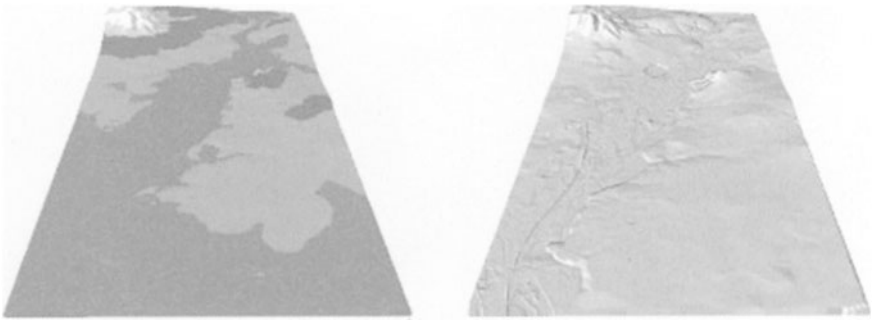


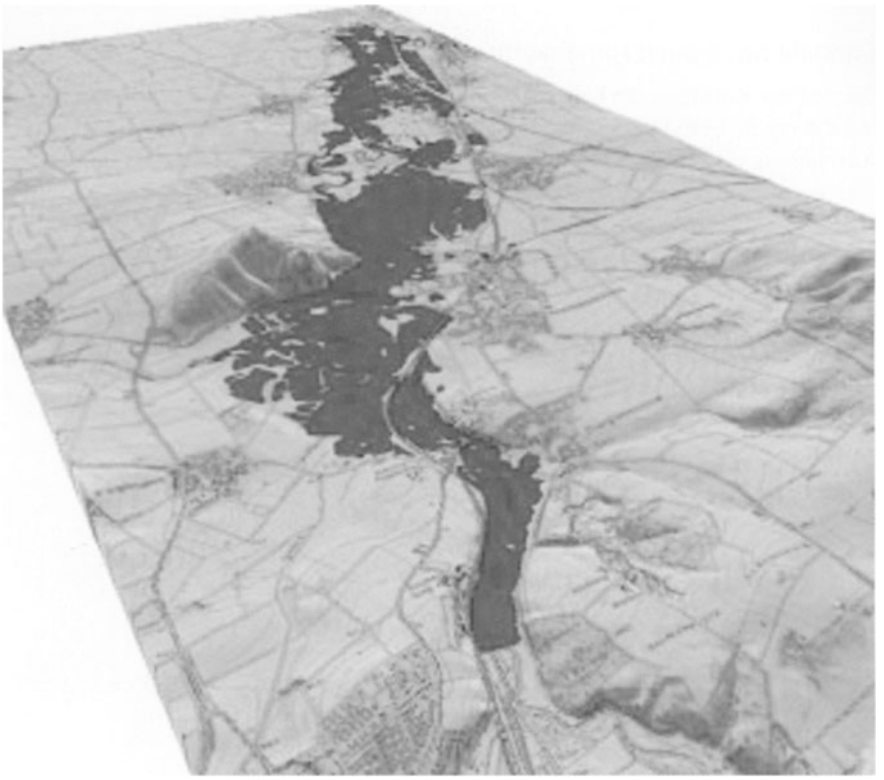
Abb. 5-7 Höhenabhängige Färbung eines Geländemodells

5.3.3 Komponenten zur Navigation, Orientierung und Exploration

Dieser Abschnitt stellt kartographische 3D-Widgets zur Navigation, Orientierung und Exploration vor. Diese Komponenten können als Grundbausteine für die Konstruktion von animierten und interaktiven kartographischen Anwendungen herangezogen werden.

Animation ist in den kartographischen 3D-Widgets als integraler Bestandteil der Interaktion des Benutzers mit dem System zu verstehen. Zum Beispiel kann der Benutzer durch direkte Manipulation des Geländemodells die Kameraeinstellung steuern. Der Benutzer markiert dazu eine Stelle im Geländemodell (den Fokuspunkt) durch Mausklick. Die Kamera wird dann auf diesen neuen Fokuspunkt ausgerichtet. Die Richtungsänderung wird animiert, um einen weichen Übergang zwischen den Bildern zu gewährleisten. Das System verfügt darüber hinaus über Möglichkeiten, automatisiert Filmsequenzen in Form von MPEG- und AVI-Dateien aufzunehmen.

Klassische Geländeüberflüge lassen sich mit Hilfe explizit erstellter Flugrouten oder durch direkte Navigation aufzeichnen.



Allen Animationen liegt die Zeitmodellierung durch Verhaltensgraphen zugrunde. Sie kann zum Beispiel dazu verwendet werden, um zeitvariante kartographische Daten zu visualisieren, wie sie zum Beispiel bei einer Überflutungssimulation vorliegen. Abb. 5-8 zeigt einen Ausschnitt einer Karte, in der Überflutungsbereiche visualisiert werden. Die zeitvariante Wasseroberfläche wurde zu einer zeitvarianten Textur umgerechnet. Diese Anwendung wird in Zusammenarbeit des Instituts für Informatik der Universität Münster und des Instituts für Kartographie der Universität Hannover entwickelt.

Abb. 5-8 Geländemodell mit Überflutungsflächen

Modellierung von Navigations- und Orientierungswerkzeugen

Alle Visualisierungssysteme stellen Funktionen zur interaktiven Kamerabewegung bereit. Ein Problem, das sich bei kartographischen wie auch allgemein bei georefe-

renzierten Daten zeigt, ist, daß der Betrachter speziellen Einschränkungen unterworfen ist, etwa bezüglich der Geländekollision oder der Aufwärtsrichtung der Kamera, die eingeschränkte Freiheitsgrade hat. Die Entwicklung geeigneter Metaphern ist eine wichtige Aufgabe, um dem Benutzer eine präzise und intuitive Navigation zu ermöglichen. Der Vorteil eines objektorientierten Herangehens an die Entwicklung von Navigationswerkzeugen liegt darin, mit vertretbarem Aufwand spezialisierte Komponenten entwickeln zu können, die als 3D-Widget in einer Widget-Bibliothek für kartographische Anwendungen eingegliedert werden können. Einige 3D-Widgets zur Navigation und Orientierung sind im folgenden kurz beschrieben.

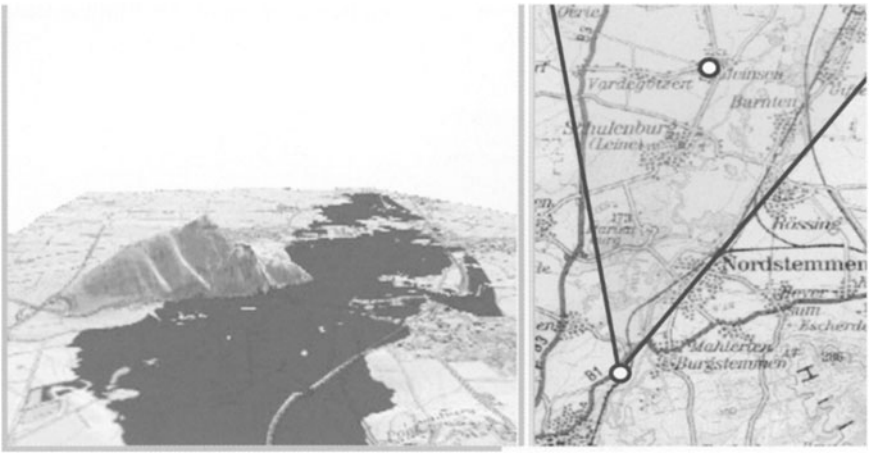


Abb. 5-9 Perspektivische Karte und Metakarte

Das *Metakarten-Widget* besteht aus einer separaten Übersichtskarte, die dem Benutzer den momentanen Standpunkt der virtuellen Kamera anzeigt und zugleich den insgesamt durch die Kamera sichtbaren Teil der Karte kennzeichnet (Abb. 5-9). Für ein Metakarten-Widget kann eine individuelle, auf die Navigationsaufgaben abgestimmte Textur verwendet werden. Im allgemeinen ist diese Textur das Resultat der Berechnung einer orthographischen Projektion des Geländemodells von oben. In der Metakarte werden häufig für das Geländemodell andere graphische Parameter und Auflösungen gewählt, weil die absolute Ausdehnung der Metakarte deutlich kleiner ist als die Ausdehnung in den 3D-Sichten auf das Geländemodell und die Orientierung mit sich verringender Distanz zwischen Beobachter und Modell verloren geht. Daher ist in Hinblick auf Generalisierungsaspekte eine besondere Gestaltung notwendig. Die direkte graphische Manipulation des Kamerastandpunktes und die Drehung des Sichtvolumens in der Metakarte und die Synchronisation der 3D-Sichten wird von Verhaltensknoten übernommen.

Landschaftsmarken-Widget

Das *Landschaftsmarken-Widget* verwaltet eine Sammlung von fachlich klassifizierten und georeferenzierten Landschaftsmarken eines Landschaftsmodells. Jede Landschaftsmarke gehört einer Landschaftsmarkenkategorie an. Landschaftsmarken lassen sich unabhängig von ihrer Kategorie hierarchisch anordnen. Eine Landschaftsmarke wird graphisch durch einen Glyphen, das heißt ein 3D-Symbol, im Geländemodell repräsentiert (Abb. 5-10). Technisch verbirgt sich hinter jeder Landschaftsmarke einerseits ein Fachobjekt (zum Beispiel verwaltet in einer Datenbank) und andererseits dessen computergraphische Repräsentation. Das Landschaftsmarken-Widget konstruiert für jede Landschaftsmarke ein Fachobjekt und legt die für die Visualisierung notwendigen Rendering-Komponenten, Geometriegraphen und Verhaltensgraphen an. Die Verhaltensgraphen implementieren zum Beispiel die interaktive Auswahl einer Landschaftsmarke durch den Benutzer und das daraufhin erfolgende automatische Heranfliegen der Kamera an die Landschaftsmarke.

Abb. 5-10 Landschaftsmarken in einem Geländemodell

Kamerasteuerungs-Widget

Das *Kamerasteuerungs-Widget* definiert die Interaktion des Benutzers mit der Kamera einer 3D-Sicht. Im allgemeinen stellen Visualisierungssysteme Steuerungsmechanismen für die Navigation in einer virtuellen Szene bereit. Für ein Geländemodell jedoch sind präzisere Mechanismen erforderlich, die berücksichtigen, daß der Betrachter nicht unter das Gelände gelangt, daß die Aufwärtsrichtung der Kamera fest vorgegeben ist und daß der Betrachter sich selbst im Geländemodell, jedoch nicht der Betrachter das Geländemodell bewegt.

Das Widget legt einen Fixpunkt in der Landschaft fest, der visuell dargestellt und interaktiv gewählt wird. Um diesen Fixpunkt kann der Benutzer mit festem Abstand

rotieren oder sich mit variablem Abstand an den Fixpunkt annähern bzw. sich entfernen. Die Wahl eines Fokuspunktes mit restringierter Bewegung erwies sich im bisherigen Arbeiten als intuitiv und effektiv benutzbar. Die momentanen Kameraeinstellungen können als Fachobjekte aufgezeichnet und in einer Projektstruktur abgelegt werden.

Überfliegungs-Widget

Das *Überfliegungs-Widget* editiert eine Flugroute für ein Geländemodell mit Hilfe einer Raumkurve. Die Kurve wird direkt im Landschaftsmodell editiert, indem die Stützpunkte in Form von Landschaftsmarken manipuliert werden. Die Überfliegung kann automatisch aufgezeichnet werden, wobei sich Einschränkungen hinsichtlich der Flughöhe (zum Beispiel feste Höhe über Grund) festlegen lassen. Technisch besteht dieses Navigationswerkzeug aus einem Widget, das die gesamte Interaktion mit den 3D-Stützpunkten übernimmt. An Rendering-Komponenten werden die Flugkurve und die Stützpunkte erzeugt. Ein Verhaltensgraph animiert die Kameraparameter.

Abb. 5-11 Höhenmessung und Weglängen-Berechnung mit einem Messungs-Widget

Modellierung von Explorationswerkzeugen

Die kartographischen 3D-Widgets zur Exploration untersuchen Fachobjekte und visualisieren die Ergebnisse solcher Untersuchungen. Wesentlich für die Genauigkeit einer Exploration ist, daß die Anfragen mit Hilfe der visualisierten Fachobjekte formuliert werden, die Analyse aber auf den eigentlichen Fachobjekten ausgeführt werden. Zum Beispiel werden bei einer Positionsabfrage durch den Benutzer die Geländekoordinaten dadurch ermittelt, das der Schnitt zwischen einem imaginären Strahl vom Blick zum selektierten Punkt und des Geländemodells mit Hilfe

des Raytracings ermittelt wird. Dabei wird auf die ursprünglichen, voll aufgelösten Daten des Geländemodells durchgeführt, um eine möglichst exakte Positionsangabe in 3D zu erhalten, die nicht vom momentanen Level-of-Detail abhängt. Im folgenden sind zwei Analyse-Widgets des vorgestellten kartographischen Visualisierungssystems skizziert.

Messungs-Widget

Das *Messungs-Widget* kann für Aufgaben der Geländemessung verwendet werden. Gemessen werden können Höhenwerte und Weglängen (Abb. 5-11). Ein Weg besteht aus einer Menge von Referenzpunkten, die durch einen Polygonzug verbunden sind. Die Berechnung der Weglänge erfolgt mit den Fachobjekten des Geländemodells und kann von einer Anwendung algorithmisch festgelegt werden. Die einzelnen Referenzpunkte werden durch interaktive Auswahl von Geländepunkten spezifiziert. Die Schnittpunkte werden durch einen 3D-Strahltest ermittelt. Jeder Weg repräsentiert zugleich ein neues Fachobjekt, das mit dem Geländemodell zusammen abgelegt werden kann. Technisch enthält ein solches Widget Rendering-Komponenten zur Visualisierung des Weges und zur Anzeige der Weglänge, einen Geometriegraphen und einen Verhaltensgraphen, der die interaktive Eingabe und Verschiebung von Stützpunkten übernimmt.

Abb. 5-12 Browser-Widget zur Analyse von Geländemodellen

Browser-Widget

Ein *Browser-Widget* führt in einem lokalen Bereich des Geländemodells anwendungsspezifische Operationen auf den Geländedaten aus. Die Operationen sind auf einen Teilbereich des Geländes begrenzt, zum Beispiel einem zylindrischen oder rechteckigen Bereich. Die Bereiche können interaktiv im Geländemodell positioniert werden. Innerhalb des Bereichs werden die Geländedaten aus dem hybriden Geländemodell ermittelt und an die Anwendung zur Analyse übergeben. Die Anwendung

kann diese Informationen zum Beispiel mit Hilfe von Datenbankanfragen auswerten. Abb. 5-12 zeigt ein Browser-Widget, das die minimale bzw. maximale Geländehöhe in einem Bereich ermittelt und alle Geländepunkte mit diesen Höhenwerten visuell darstellt. Technisch ist das Browser-Widget durch einen Geometriegraphen implementiert, der den Abfragebereich durch eine halbtransparente Ummantelung darstellt, und durch einen Verhaltensgraphen, der die interaktive Positionierung des Bereichs übernimmt.

5.4 Implementierung des kartographischen Visualisierungssystems

Die in diesem Beitrag vorgestellten Konzepte zur objektorientierten kartographischen Visualisierung wurden mit dem objektorientierten Visualisierungs- und Animationssystem *MAM/VRS* prototypisch implementiert. Dazu wurden einige speziell auf die Bedürfnisse der Kartographie und der kartographischen Animation abgestimmte neue Visualisierungskomponenten entwickelt. Abb. 5-13 zeigt die Benutzerschnittstelle des kartographischen Visualisierungssystems. Eine Darstellung der technischen Aspekte des Systems findet sich bei Döllner et al (1999).

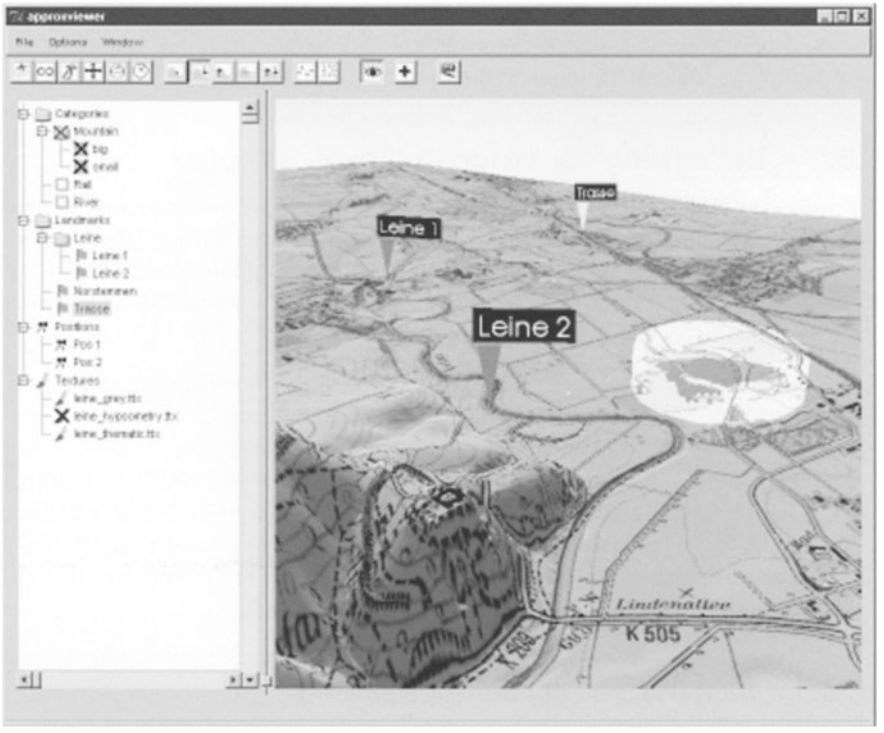


Abb. 5-13 Benutzerschnittstelle der kartographischen Visualisierungsumgebung

Das *Virtual Rendering System* VRS wurde als generische Schnittstelle herangezogen, das auf unterschiedliche 3D-Renderingsysteme, zum Beispiel *OpenGL*, *POV-Ray* und *RenderMan*, über eine einheitliche, leicht benutzbare Schnittstelle zugreift. *MAM/VRS* implementiert alle im ersten Teil beschriebenen Software-Konzepte und unterstützt insbesondere die Komponentenbildung. Zur Echtzeit-Visualisierung wird das Rendering-System *OpenGL* genutzt, da es zum einen vollständig portabel ist und zum anderen technisch innovative computergraphische Fähigkeiten bereitstellt, die zudem teilweise oder ganz von moderner Graphikhardware unterstützt wird. Insbesondere ist bei den kartographischen Widgets die Texturierung von Bedeutung. Gerade diese Funktionalität wird auch auf PC-Graphikhardware durch entsprechende Graphikkarten sehr gut unterstützt.

Die Benutzerschnittstelle wurde mit der interpretativen Skriptingsprache Tcl/Tk (Ousterhout 1994) erstellt, die eine einfache und portable Programmierung von graphischen Benutzeroberflächen ermöglicht. Diese Skriptingsprache zeichnet sich durch die ihre leichte Erlernbarkeit, Flexibilität und Erweiterbarkeit aus. *MAM/VRS* ermöglicht die Kopplung dieser Skriptingsprache mit der in der Programmiersprache C++ geschriebenen Bibliothek. Darüber hinaus können das System *MAM/VRS* und die kartographischen 3D-Widgets vollständig in diese Skriptingsprache aufgenommen werden, so daß die Entwicklung weiterer kartographischer Anwendungen in Tcl/Tk erfolgen kann. Die Wahl zwischen der C++-Schnittstelle und der Tcl/Tk-Schnittstelle bietet die Voraussetzung für den flexiblen Einsatz zur Implementierung von Anwendungen für die kartographische Visualisierung.

5.5 Bewertung der vorgestellten Konzepte

Software-Systeme für objektorientierte kartographische Visualisierung und Animation sind durch ihre hohe Komplexität charakterisiert. 3D-Rendering, 3D-Modellierung und die Komponenten-Entwicklung stellen hohe Anforderungen an den Software-Entwickler, die auf Dauer nur durch entsprechende Methoden des Software-Engineerings bewältigt werden können.

Visualisierungssysteme bieten eine Fülle technischer Fähigkeiten, jedoch ist es erforderlich, gezielt auf die Bedürfnisse der Kartographie zugeschnittene Komponenten zu entwickeln. Die Erweiterbarkeit eines Visualisierungssystems stellt insbesondere bei nicht-objektorientierten Systemen (zum Beispiel VRML) ein grundsätzliches Problem dar. Hinzu kommt, daß neue, innovative Rendering-Verfahren einen direkten Zugriff auf zugrundeliegende Rendering-Systeme (zum Beispiel *OpenGL*) erfordern, um qualitativ hochwertige, in Echtzeit rechenbare Resultate zu bekommen.

In dem vorgestellten objektorientierten kartographischen Visualisierungssystem hat sich weiter gezeigt, daß anspruchsvolle, interaktive und animierte Visualisierungen nicht den „high-end"-Workstations vorbehalten sein müssen. Der gezielte Einsatz computergraphischer Mittel, wie zum Beispiel das Texturieren, in Verbindung mit effizienten Datenstrukturen, wie zum Beispiel dem Approximationsbaum,

erlauben es, auf heutiger PC-Hardware interaktiv mit realen kartographischen Datensätzen zu arbeiten.

Es zeigt sich allerdings auch, daß die technischen Möglichkeiten allein nicht automatisch zu einer adäquaten Visualisierung kartographischer Daten führen. Zum Beispiel müssen die Texturen den Regeln der kartographischen Gestaltung entsprechen. Die Entwicklung geeigneter Metaphern in Form einer Bibliothek kartographischer 3D-Widgets schafft hier die Voraussetzung für den Vergleich, den Test und die Weiterentwicklung von allgemein einsetzbaren kartographischen Komponenten für die Visualisierung (Buziek und Döllner 1999).

Das vorgestellte objektorientierte Herangehen an die kartographische Visualisierung und Animation birgt ein großes Potential für die interaktive kartographische 3D-Darstellung. Auf der dem Buch beiliegenden CD finden sich einige Animationssequenzen und die erste Version eines Programmsystems (LandExplorer) , die die Arbeitsweise des kartographischen Visualisierungssystems veranschaulichen.

5.6 Literatur

Baumann K, Döllner J, Hinrichs K, Kersting O (1999) A hybrid, hierarchical data structure for real-time terrain visualization. In: Proceedings Computer Graphics International 1999, IEEE Computer Society, 85-92

Brown K, Petersen D (1999) Ready-to-run Java 3D™. John Wiley & Sons

Buziek G, Döllner J (1999) Concept and implementation of an interactive, cartographic virtual reality system. In: Proceedings International Cartographic Conference ´99, Vol. 1, 637-648

Buziek G, Kruse I (1992) The DTM-System TASH in an interactive environment. In: EARSeL Advances in Remove Sensing, 1(3):129-134Cunningham S, Craighill NK, Fong MW, Brown JR (1992) Computer graphics using object-oriented programming. John Wiley & Sons

Döllner J, Hinrichs K (1999) A generic application programming interface for 3D graphics. Bericht Nr. 13/99-I, Angewandte Mathematik und Informatik, Universität Münster

Döllner J, Kersting O, Hinrichs K, Baumann K (1999) Konzepte und 3D-Visualisierung interaktiver, perspektivischer Karten. In: Strobl/Blaschke (Hrsg.), Angewandte Geographische Informationsverarbeitung XI. Beiträge zum AGIT-Symposium Salzburg 1999. Wichmann-Verlag, 128-139

Döllner J, Hinrichs K (1998) Interactive, animated 3D widgets. In: Proceedings Computer Graphics International 1998, IEEE Computer Society, 278-286

Döllner J, Hinrichs K (1997) Object-oriented 3D modelling, animation, and interaction. In: The Journal of Visualization and Computer Animation, 8(1):33-64, http://wwwmath.uni-muenster.de/~mam

Enzmann A, Kretzschmar L, Young C (1994) Ray tracing worlds with POV-Ray. Waite Group Press

Foley J, van Dam A, Feiner S, Hughes J, Phillips R (1994) Introduction to computer graphics. Addison-Wesley

Gamma E, Helm R, Johnson R, Vlissides J (1996) Design patterns: elements of reusable object-oriented software. Addison-Wesley

MacEachren AM (1994) Visualization in modern cartography: setting the agenda, In: MacEachren AM, Taylor F (eds) Visualization in Modern Cartography. Pergamon Press, 1-12

McCarthy M, Carty A (1998) Building 3D worlds in Java and VRML. Prentice Hall

Microsoft (1999) Direct X Web Site. http://www.microsoft.com/directx

Mohan S (1998) The fourth generation of 3D graphics APIs has arrived. Sun Microsystems, Inc., White Paper

Oestereich B (1997) Objekt-orientierte Softwareentwicklung mit der Unified Modeling Language UML. 3. Auflage, Oldenbourg, München

Ousterhout J (1994) Tcl and the Tk Toolkit. Addison-Wesley

Schroeder W, Martin K, Lorensen B (1998) The visualization toolkit. An object-oriented approach to computer graphics, 2nd ed. Prentice-Hall

Upstill S (1990) The RenderMan Companion. A programmer's guide to realistic computer graphics. Addison-Wesley

Ward G, Shakespeare R (1998) Rendering with Radiance. The art and science of lighting visualization. Morgan Kaufmann Publishers

Wernecke J (1994) The Inventor Mentor. Programming object-oriented 3D graphics with Open Inventor, Release 2. Addison-Wesley

Woo M, Neider J, Davis T (1997) OpenGL Programming Guide, 2nd ed. Addison-Wesley